Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Benko

Simulacija pretoka krvi po vratnih žilah

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Saša Divjak
Somentor: doc. dr. Matija Marolt

Ljubljana, 2017

To diplomsko delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva-Deljenje pod enakimi pogoji 2.5 Slovenija.* To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, dajejo v najem, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani `http://creativecommons.si/` ali na Inštitutu za intelektualno lastnino, Streliška 1, 1000 Ljubljana.

Izvorna koda diplomskega dela, njenih rezultatov in v ta namen razvite programske opreme je ponujena na naslovu `https://github.com/PeterBenko/WebVeins` pod GNU General Public License, različica 3. To pomeni, da se lahko prosto uporablja, distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani `http://www.gnu.org/licenses/`.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.
Slike so izdelane s pomočjo jezika PGF/TikZ.
Psevdokoda izdelana s knjižnico ALGORITHMICX
Binarni diagrami izrisani z knjižnico BYTEFIELD

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preučite področje simulacije pretokov fluidov po tridimenzionalnih modelih. Analizirajte odprtokodne knjižnice, ki vsebujejo simulacijske algoritme, in integrirajte najprimernejšo knjižnico z ogrodjem za spletno vizualizacijo medicinskih podatkov Med3D.

Zahvaljujem se doc. dr. Matiji Maroltu, prof. dr. Saši Divjaku in še posebno asist. dr. Cirilu Bohaku za pomoč pri izdelavi in pisanju diplomskega dela. Posebna zahvala gre tudi moji partnerki Gaji za podporo, motivacijo in slovnično pomoč. Za podporo in vzpodbudo bi se rad zahvalil tudi svojim staršem.

Kazalo

Povzetek

Naslov: Simulacija pretoka krvi po vratnih žilah

Avtor: Peter Benko

Cilj te diplomske naloge je pregled širokega področja obstoječih rešitev za simulacijo pretoka fluida po tridimenzionalnem modelu, uporaba najprimernejše v spletni aplikaciji, z vizijo, da se jo vpne v obstoječi program Med3D. Simulacijo želimo pognati po modelu žile, pridobljenim iz postopka sestavljanja medicinskih slik v programu NeckVeins. V nalogi v grobem spoznamo tudi fizikalno ozadje načina simulacije po mrežni Boltzmannovi metodi, na kateri sloni knjižnica, ki smo jo izbrali po pregledu področja. Spoznamo načine zapisov podatkov, ki so potrebni za uspešno povezovanje vmesnikov obstoječih programov. Predstavljena je končna implementacija programa, ki smo ga razvili, algoritmi, uporabljeni za pomoč pri obdelavi modela, ter odločitve pri snovanju programa.

Ključne besede: Palabos, simulacija pretoka fluidov, JavaScript, Mrežna Boltzmannova metoda, MBM.

Abstract

Title: Blood flow simulation in neck veins

Author: Peter Benko

The goal of this thesis is an overview of the vast field of existing computational fluid dynamics solvers for a three-dimensional model and integration of the most suitable into a web application, with the vision to integrate it into the existing Med3D program. The simulation is to be run through a blood vessel, that was generated by assembling medical images in NeckVeins. In the paper we get briefly acquainted with the physical background of the Lattice Boltzmann method, on which the chosen solution from the overview of existing solutions is based on. We inspect the file formats, that are needed for a successful integration of different interfaces, that the existing software already defines. Presented is also the final implementation of the developed program, the algorithms used for supporting the preparation of the model for the simulation, and design decisions met during the development.

Keywords: Palabos, computational fluid dynamics, CFD, JavaScript, lattice Boltzmann method, LBM.

Poglavje 1

Uvod

Dandanes so računalniki prisotni povsod v našem življenju. Uporabljajo se tako v zasebne kot tudi profesionalne namene. Z njihovim razvojem se odpirajo vedno nova področja uporabe. Z napredkom v procesorski in spominski zmogljivosti grafičnih kartic sta nam tako omogočena obdelava in prikaz vedno večje količine podatkov.

Eno izmed področij, kjer vizualni prikaz podatkov hitro pridobiva na veljavi, je medicina. Ker je predmet opazovanja pogosto premajhen ali pa nedostopen za ogled s prostim očesom, se pogosto uporablja alternativne načine za prikaz telesa in njegove notranjosti, kot sta na primer slikanje z magnetno resonanco (angl. magnetic resonance imaging; MRI) in rentgensko slikanje s pomočjo računalniške tomografije (angl. computerized axial tomography scan; CT (CAT) scan). Rezultati teh posegov so v osnovni obliki zajeti kot dvodimenzionalne slike, ki si po tretji dimenziji sledijo v rezinah (angl. slice) določene debeline. Posledično so podatki v takšni obliki zahtevni za analizo. Za lažjo prostorsko predstavo se s pomočjo metod uporabne matematike in računalniških znanosti iz zajetih slik rekonstruira 3D model opazovanega objekta. Ob rekonstrukciji se lahko osredotočimo tudi na izključno eno lastnost zajetih podatkov, pri MRI slikah na primer na odtenke sivine, torej na posamezno vrsto tkiva. V primeru programa NeckVeins se osredotočamo na prepoznavanje in rekonstrukcijo žil. Pridobljeni model žile

ni priročen zgolj za vizualizacijo, ampak je ob primerni kvaliteti zajema primeren tudi za izvajanje simulacije pretoka krvi.

Simulacije računalniške dinamike fluidov (angl. computational fluid dynamics; CFD) so že nekaj let stalnica v strojniškem svetu, kjer se uporabljajo v namene analize aerodinamike predmetov ali analize pretoka tekočin v ceveh, črpalkah ipd. Na področju medicine je simulacija pretokov še neuveljavljeno področje. Kljub množici programskih rešitev za področje strojništva za področje medicinskih simulacij pretokov takšnih rešitev ni veliko.

Kardiovaskularna simulacija nam omogoča zgodnje odkrivanje nepravilnosti v pretoku krvi, kot na primer vrtinčenje ter akutno zvišanje pritiska na stene žil, kar nakazuje na nevarnost anevrizme. Ker je simulacija pretoka krvi po celotnem krvožilju računsko prezahtevna, se pri analizi osredotočamo na omejeno področje in na omejeno natančnost krvožilja (na primer na model, kjer žile z manjšim premerom niso vključene).

Končni cilj moje diplomske naloge je pognati simulacijo pretoka krvi po modelu žile znotraj programa NeckVeins. V okviru diplomskega dela smo pregledali obstoječe programe in knjižnice za simulacijo pretoka tekočin ter izbrali najprimernejšo rešitev za naš primer. Izbrano implementacijo smo vpeli v obstoječi program.

Poglavje 2

Teoretično predznanje

Za smiselno analizo in uspešno izbiro simulatorja pretoka tekočin je potrebno vsaj grobo poznavanje fizikalne problematike, ki jo programje rešuje. Prav tako se v izvorni kodi knjižnic pogosto pojavljajo reference na fizikalno ozadje, pri čemer nam predznanje izjemno olajša branje in uporabo že obstoječe kode. Zaradi kompleksnosti in širine teoretične podlage se bomo omejili zgolj na osnove.

2.1 Tekočina kot zvezna snov

Fluid, če ga opazujemo na nivoju "makro sveta", si lahko predstavljamo kot enotno maso, ki v celoti zapolnjuje prostor, v katerem se nahaja. Gibanje takšne mase opisujemo s parcialnimi diferencialnimi enačbami. Te enačbe so rešljive zgolj z vpeljavo določene ravni diskretizacije in približkov, ki pa lahko spremenijo samo obliko parcialnih diferencialnih enačb. Te nove enačbe, ki se jih z numeričnimi procesi dejansko rešuje, se pogosto imenuje prirejene diferencialne enačbe. V glavnem poznamo tri takšne metode reševanja parcialnih diferencialnih enačb:

- metodo končnih diferenc
- metodo končnih volumnov

- metodo končnih elementov

2.1.1 Eulerjeve in Navier-Stokesove enačbe

Navier-Stokesove enačbe definirajo zvezni sistem nelinearnih parcialnih diferencialnih enačb, ki opisujejo ohranjanje mase, energije in momenta fluida. Za newtonski fluid se jih lahko zapiše kot:

$$\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} = 0 \tag{2.1}$$

, pri

$$Q = \begin{bmatrix} \rho \\ \rho u \\ e \end{bmatrix}, E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ u(e+p) \end{bmatrix} - \begin{bmatrix} 0 \\ \frac{4}{3}\mu\frac{\partial u}{\partial x} \\ \frac{4}{3}\mu u\frac{\partial u}{\partial x} + \kappa\frac{\partial T}{\partial x} \end{bmatrix} \tag{2.2}$$

kjer je

ρ gostota fluida

u hitrost

e skupna energija na prostorsko enoto

p pritisk

T temperatura

μ koeficient viskoznosti

κ toplotna prevodnost

Skupna energija e vsebuje notranjo energijo na prostorninsko enoto $\rho\epsilon$ (kjer je ϵ notranja energija na enoto mase) in kinetično energijo na enoto volumna $\frac{\rho u^2}{2}$. Te enačbe morajo biti uravnotežene tako s povezavo med μ, κ in stanjem fluida, kot tudi z neko enačbo stanja, kot je na primer splošna plinska enačba.

Mnogi pretoki, ki nas zanimajo, so stabilni (neodvisni od časa) oziroma jih lahko kot takšne vsaj obravnavamo. Za takšne tokove nas zanima zgolj

stabilna rešitev Navier-Stokesovih enačb, in ne rešitev vmesnega obdobja prehodnih stanj. Stabilna rešitev enodimenzionalnih Navier-Stokesovih enačb mora zadovoljiti

$$\frac{\partial E}{\partial x} = 0 \tag{2.3}$$

Če zanemarimo viskoznost in toplotno prevodnost, tako dobimo Eulerjeve enačbe. V dvodimenzionalnem kartezijskem koordinatnem sistemu se jih lahko zapiše kot

$$\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} = 0 \tag{2.4}$$

, pri

$$Q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}, E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \end{bmatrix}, F = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(e+p) \end{bmatrix} \tag{2.5}$$

, kjer sta u in v kartezijski komponenti za hitrost. [14]

2.2 Fluid kot diskretna snov

Fluid si lahko predstavljamo tudi kot diskreten prostor atomov in molekul, med katerimi je prazen prostor. Tako imenovani "mikro svet" se opisuje z molekularno dinamiko. Primera opisov takšnega pogleda na fluid sta med drugim metoda diskretnih elementov in kinetična teorija plinov, ki razlaga makroskopske lastnosti fluida, obrazložene z lastnostmi molekularnega sestava in gibanja.

2.2.1 Kinetična teorija plinov

Kinetična teorija plinov je za nas pomembna predvsem zato, ker prav na njej sloni model obnašanja delcev v Bolztmannovi enačbi [9]. V njej je mikroskopsko stanje fluida opisano z Newtonskim modelom, ki določi natančen položaj in hitrost vsake posamične molekule. Same molekule so pogosto definirane kot rigidne krogle, ki druga na drugo delujejo s popolnoma elastičnimi

trki. Tako okrnjen opis interakcije pa ni obvezna omejitev, saj je možno trke opisati tudi z bolj prefinjenim kolizijskim modelom. Sama kinetična teorija plinov pa bi bila za računalniško simulacijo prezahtevna, saj je že v eni sami enoti fluida $6,02 * 10^{23}$ delcev, katerih obnašanje bi morali simulirati, zato je za skalabilno simulacijo na molekularni ravni potrebna določena posplošitev.

2.3 Mrežna Boltzmannova metoda

Začetki mrežne Boltzmannove metode segajo v osemdeseta leta prejšnjega stoletja. V tistem času so se razvili mrežni plinski celični avtomati, posebni celični avtomati, namenjeni prav simulaciji pretoka fluidov.

2.3.1 Celični avtomati

Celične avtomate po definiciji predstavlja n-dimenzijska mreža, v kateri vsaka celica nosi eno od vnaprej določenih stanj. Vsaka celica ima definirano začetno stanje, ki pa se spreminja skozi diskretne časovne inkremente simulacije. Vsaka sprememba predstavlja novo generacijo. Kako se nove generacije ustvarjajo pa je odvisno od vnaprej določene funkcije, ki iz stanja celice in njene okolice določi novo stanje celice. Ta funkcija je tipično enotna za vse celice v mreži. Najbolj znan primer celičnega avtomata je Conwayeva igra življenja [12].

2.3.2 Mrežni plinski celični avtomati

Mrežni plinski celični avtomati so specializirani celični avtomati, namenjeni simulacijam pretoka fluidov. Najbolj poznan izmed teh modelov se imenuje FHP, ki se imenuje po izumiteljih iz leta 1986[11]: Uriel Frisch, Brosl Hasslacher in Yves Pomeau. Model poizkuša posnemati fizikalne lastnosti plina s tem, da vsili idealizirano obliko plina, katerega molekule se nahajajo na diskretni heksagonalni mreži. V tem modelu so celice avtomata vozlišča mreže, na katerih se molekule nahajajo, potujejo pa lahko zgolj po šestih povezavah

v sosednja vozlišča. Model ima tako šest ali sedem možnih hitrosti, odvisno od variacije modela. Modela FHP-II in FHP-III vpeljeta dodatno "mirujoče" stanje (na diagramu 1 označeno kot e_0), ko delec ostane v vozlišču in se ne premakne, vendar lahko povzroči kolizije v naslednji iteraciji [3].

Nove generacije v tem avtomatu nastajajo v dveh korakih. Na začetku vsake iteracije se molekule premaknejo po povezavah v eno od sosednjih vozlišč. Premikom sledi preračunavanje trkov, ki molekule na novo prerazporedi po povezavah, odvisno od načina vstopa v vozlišče. Trki se obravnavajo po pravilu elastičnih trkov, ki morajo ohranjati maso in gibalne količine. Ker imajo trki v FHP modelu v določenih primerih več možnih rešitev, so pa nedeterministični, se za reševanje takšnih situacij uporablja generator psevdo naključnih števil za izbiro enega od možnih izidov.

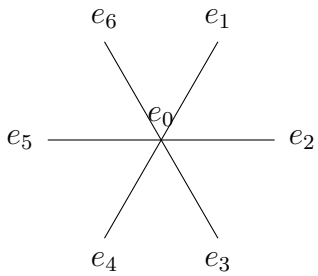


Diagram 1: Hexagonalna mreža uporabljena v FHP modelu, označena z vozlišči stanj

Presenetljivo lahko ta preprost način popolnoma reproducira fizikalno obnašanje nestisljivega toka fluida, kar pomeni, da je, če uporabimo dovolj veliko število molekul, z njim možno na makroskopski ravni rešiti Navier-Stokesove enačbe [31].

FHP model pa ima tudi pomanjkljivost. Zaradi oblike mreže je zelo težko prevedljiv v tridimenzionalni svet, kar se rešuje z uvedbo četrte, navidezne dimenzije, kar pa naredi izračun zelo kompleksen [18]. Izkazalo se je tudi, da so počasni za izračune diskretnih stanj in veliko bolj učinkoviti v aritmetiki s plavajočo vejico [9]. Prav tako trpi model za numeričnim šumom, kar pomeni, da je za dosego sprejemljive kovergence potrebno v model vnesti

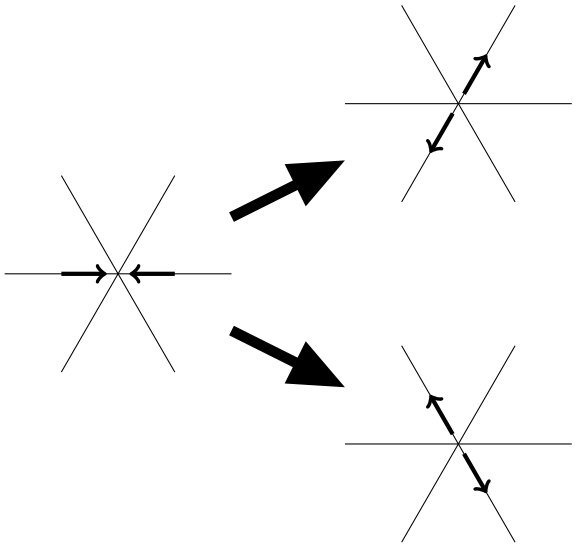


Diagram 2: Prikaz čelnega trka z obema možna izidoma

izjemno velike količine molekul in potem rezultate še povprečiti, kar nam izračun še dodatno otežuje.

2.3.3 Mrežna Boltzmannova metoda

Mrežna Boltzmannova metoda je metoda neke vrste srednjega, "mezo" sveta, saj predstavlja vez med obema svetovoma. Kot naslednica mrežnih plinskih celičnih avtomatov se je pojavila v zadnjem dobrem desetletju. Probleme predhodnika premaguje tako, da je zamenjala diskretna molekularna stanja za točne opise, ki temeljijo na statističnih lastnostih molekul fluida, pridobljenih iz statistične mehanike in funkcije gostote porazdelitve [19]. Ta funkcija opisuje verjetnostno gostoto, da je molekula fluida na določeni točki v prostoru z določeno hitrostjo. Odvisna je od sedmih spremenljivk: tri za pozicijo v prostoru (x, y, z), tri za hitrost v prostoru (ξ_x, ξ_y, ξ_z) ter eno za čas (t). Funkcija gostote porazdelitve ($f(x, \xi, t)$) je zelo direktno povezana z makroskopskimi spremenljivkami, ki smo jih že navajeni: gostota (2.6), kinetična energija (2.7), notranja energija (2.8), itd.

$$\rho(\vec{x},t) = m\int_{-\infty}^{\infty} f d^3\xi \tag{2.6}$$

$$\rho\vec{u} = m\int_{-\infty}^{\infty} \vec{\xi} f d^3\xi \tag{2.7}$$

$$\rho\epsilon = m\int_{-\infty}^{\infty} (\vec{\xi}-\vec{u}) f d^3\xi \tag{2.8}$$

Očitno je, da se moramo za prevod porazdelitvene funkcije na makroskopski svet znebiti odvisnosti od hitrosti, saj so običajne makroskopske spremenljivke odvisne zgolj od prostora in časa. Da to dosežemo, odvisnost od hitrosti odintegriramo stran. Tako je na primer makroskopska gostota fluida nič drugega kot preprost integral porazdelitvene funkcije nad celotnim hitrostnim prostorom.

Opazimo, da ostane odvisnost od položaja v prostoru med pretvorbo porazdelitvene funkcije na makroskopske enačbe nespremenjena. To pomeni, da je porazdelitvena funkcija hitrosti zvezna spremenljivka v tradicionalnem pomenu besede, in predstavlja tok na isti ravni kot tradicionalne makroskopske numerične metode. To opažanje ovrača eno od najbolj razširjenih napačnih predstav o mrežni Boltzmannovi metodi, in sicer, da je zasnovana zgolj za reševanje mikroskopskih fizikalnih problemov. Ko za mrežno Boltzmannovo metodo rečemo, da je mikro- ali mezoskopska metoda, se s tem nanašamo predvsem na njeno teoretično ozadje in na vrsto enačb, ki jih rešuje, in ne na ločevanje med diskretnim in zveznim pristopom opisovanja problema.

Dinamika prostora in časa v porazdelitveni funkciji je ponavadi podana s tako imenovano Boltzmannovo enačbo (2.9). Prav tako kot Navier-Stokesove enačbe, ki so uravnotežene enačbe momenta fluida, je Boltzmannova enačba uravnotežena enačba gostote delcev. Lahko si predstavljamo neskončno majhen kontrolni prostor. Da se število delcev v tem prostoru poveča, mora vanj vstopiti delec, ki je vanj pripotoval v ravni črti skozi prazen prostor ali pa se je v kontrolni prostor odbil po trku z drugim delcem.

Leva stran Boltzmannove enačbe opisuje vstop delca po prepotovani pro-

$$\left.\frac{Df}{Dt}\right|_{Transport} = \left.\frac{Df}{Dt}\right|_{Trk}$$
$$\frac{\partial f}{\partial t} + \xi_\alpha \frac{\partial f}{\partial x_\alpha} = C(f) \tag{2.9}$$

Slika 2.1: Boltzmannova enačba, kjer $C(f)$ modelira trk med dvema paroma delcev

sti poti, rešuje pa se jo z Lagrangevim odvodom vzdolž karakteristične smeri. Desna stran, ki opisuje trke med dvema delcema, je nekoliko bolj zapletena in je odvisna od prečnega prereza razpršitve molekul. Da lahko opišemo ta del enačbe moramo vedeti, kako na moment molekul, ki trčita, vplivajo njune začetne lastnosti pred trkom. Že za zelo preprost, popolnoma elastični trk dveh trdih krogel, je ta opis izjemno zahteven, in se ga ponavadi poenostavlja, da lahko dobimo ponazoritev potrebne makroskopske fizike pri dovolj kratkem računskem času. Izbira kolizijskega modela je izjemno pomembna, saj popolnoma določa vsebino makroskopske fizike, ki jo model ustvari.

2.3.4 Diskretizacija Boltzmannove enačbe

Da porazdelitveno funkcijo v prostoru hitrosti diskretiziramo, jo razvijemo v posplošeno Fourierovo vrsto. To naredimo s pomočjo Hermitovih polinomov (enačba 2.10). Razlog za to izbiro je, da so pri razvoju polinomov koeficienti (v enačbi 2.10 označeni z $a^{(n)}$) identični običajnim makroskopskim spremenljivkam. Tako ni na primer koeficient 0-tega reda nič drugega kot gostota fluida v določeni točki prostora in časa.

$$f(\vec{x}, \vec{\xi}, t) \approx f^N(\vec{x}, \xi, t) = \omega(\vec{\xi}) \sum_{n=0}^{N} \frac{1}{n!} H_\alpha^{(n)}(\vec{\xi}) a_\alpha^{(n)}(\vec{x}, t) \tag{2.10}$$

Opazimo, da moramo za vsak koeficient razvitega Hermitovega polinoma oz. makroskopske spremenljivke izračunati integral v prostoru hitrosti. Da

lahko te integrale rešimo na računsko cenovno ugoden način, uporabimo tehniko, imenovano Gauss-Hermitovo kvadraturno pravilo [18]. Integral (2.11) tako lahko nadomestimo z vsoto določenega števila točk pod pogojem, da poznamo vrednosti polinoma $r(\xi)$, pri stopnji, ki je manjša ali enaka $2n-1$. Pri tem je n število diskretiziranih hitrosti ξ_i, ki jih želimo opazovati, ω_i so uteži (2.12), ξ_i pa matematično ničle Hermitovih polinomov, ki v mrežni Boltzmannovi enačbi predstavljajo diskretizirane hitrosti delcev. Te hitrosti lahko imenujemo kar smerni vektorji [18]. Odvisno od izbire količine možnih vektorjev, lahko tako dobimo različne opazovalne prostore:

$$\eta = \int_{-\infty}^{\infty} \omega(\xi) r(\xi) dx = \sum_{i=1}^{n} \omega_i e(\xi_i) \tag{2.11}$$

$$\omega_i = \frac{n!}{(nH_{n-1}(\xi_i))^2} \tag{2.12}$$

Za dve točki diskretizacije dobimo numerično shemo, imenovano D1Q2. V notaciji DdQq predstavlja d število dimenzij in q število smernih vektorjev v mreži, kar so efektivno možne smeri gibanja delca. Če želimo enačbo 2.11 razširiti na več dimenzij, uporabimo Gauss-Hamiltonovo kvadraturno pravilo najprej v vsako od koordinathih smeri hitrosti, vrednosti uteži pa zmnožimo.

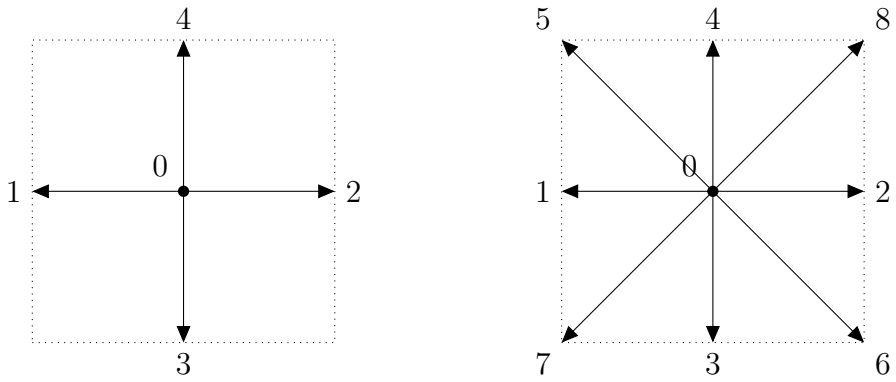


Diagram 3: Numerični shemi D2Q5 (levo) in D2Q9 (desno)

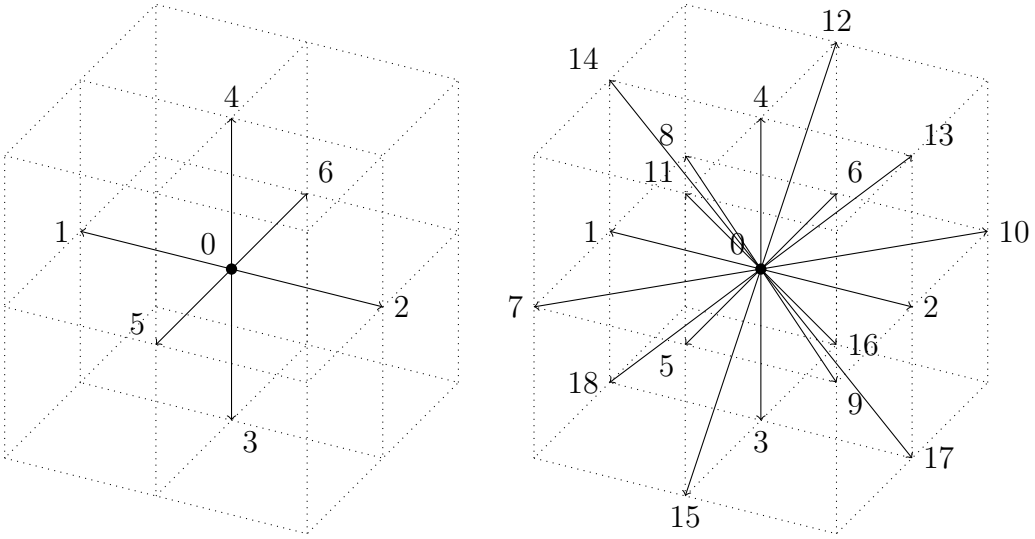


Diagram 4: Numerični shemi D3Q7 (levo) in D3Q19 (desno)

Poglavje 3

Pregled področja

Pri iskanju primerne razvojne knjižnice smo določili naslednje kriterije:

- podpirati mora datoteke v obliki, ki jih ustvari aplikacija NeckVeins
- biti mora brezplačna
- biti mora odprtokodna

Podpora za datoteke .obj in .mkd, ki jih proizvede NeckVeins, se je izkazala za najmanj omejujočo, saj je tudi za knjižnice, ki omenjenih podatkovnih oblik ne podpirajo, možno obstoječi zapis prevesti v obliko, ki jo knjižnica potrebuje. Za veliko bolj omejujoči sta se izkazali preostali dve zahtevi: odprtokodnost in brezplačnost.

V nadaljevanju sledi širši izbor aplikacij za simulacijo fluidov.

3.1 X-flow

X-Flow[37] je plačljiv programski sistem, ki se posveča reševanju problemov, kjer se opazovani predmeti med simulacijo gibajo. Trži se kot samostojna enota, ki nudi celoten postopek od uvoza modela do simulacije in končno tudi analize rezultatov. Zanimiv je tudi zaradi tega, ker lastno sposobnost paralelizacije računanja trži z možnostjo najema računalniškega oblaka za omogočanje reševanja zahtevnejših problemov.

3.2 Solidworks, Autodesk, ANSYS

Programski paketi, pogosto uporabljeni na področju strojništva[29][2][1]. Večina jih nudi poskusne študentske različice programov, vendar so v osnovi vsi plačljivi.

Omogočajo celovito računalniško podporo tako pri konstruiranju in analizi modelov, kot tudi pri načrtovanju in optimizaciji same izdelave načrtovanih objektov. Del celovitega postopka proizvodnje so tudi reševalniki pretoka fluidov, ki omogočajo simulacijo tekočine po ceveh, zraka okoli vozečega vozila, lahko pa tudi vetra v mestu [27].

3.3 Wind-US

Wind-US[35] je računska platforma razvita s strani zveze NPARC [20](National Program for Applications-Oriented Research in CFD), ki je pod okriljem NASE (National Aeronautics and Space Administration), in posledično Združenih držav Amerike. Pri razvoju je tesno sodelovala družba Boeing, ki je tudi glavni izmed zastopnikov programa. Wind-US podpira reševanje Eulerjevih in Navier-Stokes enačb za mehaniko fluidov, poleg tega pa vsebuje tudi enačbe za reševanje turbulentnih tokov in tokov s kemičnimi reakcijami. Čeprav je program brezplačen in odprtokoden, je, ker je razvit s strani ameriške državne ustanove, na voljo zgolj ameriškim ustanovam [36].

3.4 Unicorn

Projekt Unicorn[32] je del projekta FEniCS [7], ki ga ustvarjajo tako mnoge svetovne univerze kot tudi raziskovalni inštituti. Razvit je s strani univerze in razvojnega inštituta na švedskem, natančneje Laboratorija za računsko tehnologijo (CTL - Computational Technology Laboratory) na KTH kraljevem inštitutu za tehnologijo v Stockholmu (KTH Royal Institute of Technology in Stockholm). Žal je bila zadnja aktivnost projekta Unicorn na njihovem

javnem repozitoriju 25. oktobra 2011 s predvideno naslednjo izdajo 31. decembra 2011, ki pa ni bila objavljena. V tem času Unicorn tudi ni več kompatibilen z najnovejšo različico projekta FEniCS [33]

3.5 FELiScE

FELiScE[6] (okrajšava za Finite Elements for LIfe SCiences and Engineering, Končni elementi za življenjske znanosti in inženirstvo) je projekt ekipe REO[25] s Francoskega nacionalnega inštituta za računalništvo in uporabno matematiko, Inria[15] (fr. L'Institut national de recherche en informatique et en automatique). Ekipa se posveča predvsem numeričnim simulacijam bioloških tokov in srčni elektrofiziologiji. Poleg tega so tudi člani projekta Modeling ventricle, ki se trudi ustvariti natančno simulacijo srca s samo enim srčnim prekatom. S tem bi olajšali odločitve pri načrtovanju posega na pacientih. Ker gre za prirojeno napako srca, so pacienti dojenčki ali majhni otroci, pri katerih je natančna in hitra operacija ključnega pomena. Na straneh ekipe REO je zaslediti, da bo projekt FELiScE odprtokoden, vendar so na povpraševanje po dostopu žal odgovorili odklonilno, ker v takratnem stanju projekta niso želeli predajati kode.

3.6 Sailfish

Sailfish[26] je zanimiv eksperimentalen projekt reševalnika dinamike fluidov, saj je v nasprotju z ostalimi programi, napisanimi v močno tipiziranih jezikih, kot je na primer `C++`, napisan v Pythonu. Po besedah razvijalcev zahvaljujoč implementaciji programskega ogrodja OpenCL, razvitega pod okriljem skupine Khronos, ki razvija tudi OpenGL in WebGL, ter nVidiinega programskega ogrodja CUDA C, šibkosti šibko tipiziranega programskega jezika kljub zahtevni nalogi ne pridejo do izraza.

Zdi se, da projekt sloni predvsem na enem samem razvijalcu, ki pa je, in posledično tudi javni repozitorij projekta, od začetka leta 2014 vedno manj

aktiven. V tem času je žal tudi povezava na glavni strani do dokumentacije projekta neobstoječa.

3.7 OpenFOAM

Začetki OpenFOAM-a[21] so že v osemdesetih letih zadnjega stoletja v obliki licenčnega projekta FOAM. Konec leta 2004 so glavni člani ustanovili podjetje OpenCFD z namenom nadaljnjega popolnoma odprtokodnega razvoja (dotedanji program je imel odprto dostopen le del kode) pod imenom OpenFOAM, katerega prva verzija je bila objavljena kmalu po tem. Skozi leta se je ta razvil v verjetno najbolj razširjen in poznan odprtokodni numerični reševalnik. Čeprav sta leta 2014 dva od treh ustanovnih članov družbo OpenCFD zapustila, še vedno prispevata k razvoju programa. Projekt je še vedno izjemno aktiven, tako je v letu 2015 doživel več kot tisoč doprinosov (commitov) h kodi.

Poleg dolge zgodovine in aktivnega razvoja se projekt lahko pohvali tudi z zelo obširno dokumentacijo in mnogimi vodiči in predavanji, organiziranimi s strani skupnosti. Za ogled rezultatov simulacij nudi lasten vmesnik za zagon prav tako odprtokodnega programa ParaView (poglavje 4.2).

Za ceno svoje izjemne prilagodljivosti in široke uporabe pa je program nekoliko težje uporaben kot preprosta knjižnica k programu, saj zahteva dobro mero predobdelave in ročnega vnosa parametrov, specfičnih za konkretno simulacijo.

3.8 Palabos

Palabos[22] je odprtokodni produkt podjetja FlowKit[8], ki je bilo ustanovljeno s strani raziskovalcev z Zveznega inštituta za tehnologijo v Luzani in Ženevske univerze po večletnem razvoju aplikacije. Univerza v Ženevi je tudi glavni razvijalec Palabosovega jedra in uporabljenih teoretičnih znanj.

Vezano na svoj prosto dostopni produkt podjetje trži svetovanje, izo-

braževanje in nudi tudi podporo razvijalcem. Program je kljub trženju izobraževanja dobro dokumentiran in poleg skupka zelo zanimivih in kompleksnih primerov uporabe nudi tudi obširno zbirko pripravljenih učnih primerov, ki so izjemno dobro dokumentirani in omogočajo enostaven uvod razvijalca v delo z orodjem. Poleg dokumentacije samega programa pa Palabos nudi tudi pregled uporabljene mrežne Boltzmannove metode reševanja mehanike fluidov. Zadnja izdaja programa, ki je vsebovala tudi začetke razvoja Javanske ovojnice programja, je bila 16. januarja 2015. Zadnja aktivnost na Palabosovem javnem repozitoriju [23] pa je bila dobra dva meseca kasneje. Žal od takrat projekt dozdevno miruje, vendar nas aktivnost razvijalcev na socialnih omrežjih navdaja z upanjem, da se razvoj nadaljuje.

3.9 SimVascular

SimVascular[28] je zelo mlad in obetaven program, ki se od ostalih, prej omenjenih programov, razlikuje v tem, da je bil razvit s konkretnim ciljem kardiovaskularne simulacije pretoka fluidov. Prva verzija je bila javno objavljena 28. aprila 2015. Od takrat je v neprestanem razvoju. Projekt je finančno podprt s strani ameriškega Državnega znanstvenega združenja (NSF - National Science Foundation), k razvoju pa prispevajo tudi mnoge ameriške univerze.

Program nam je ponujen kot uporabniški vmesnik, ki nudi celoten proces od obdelave zajetih medicinskih slik do simulacije pretoka po žilah. Nudi široko paleto izboljšave zajetih grobih podatkov, kar nam olajša nadaljnji potek, vpliva pa seveda tudi na kvaliteto kasnejših rezultatov. Sledi geometrično modeliranje, kjer mora uporabnik sam slediti izbranemu obrisu, pri čemer mu program pomaga z razpoznavanjem verjetnih robov. Okoli izbrane poti se tako sproti ustvarja nekakšno tridimenzionalno ogrodje, ki pa se ga lahko tudi naknadno pregleda in prilagodi. S tem ima program vse potrebne podatke za ustvarjanje modela. Za ta postopek program nudi dve različni jedri: Parasolid (uporabljen tudi v ANSYS in SolidWorks) in

PolyData. Obe možnosti iz prej definiranih struktur ustvarita primeren tridimenzionalni model. Za tem sledi mreženje modela. Za vsakega od načinov ustvarjanja modela je priporočen drug način: za Parasold model MeshSim, za PolyData model pa TetGen. Sledi še določanje vhodnih in izhodnih pogojev, specifikacija tekočine, ročna konfiguracije reševalnika (med drugim tudi izbor procesorjev, ki naj jih uporabi), in simulacija se lahko izvede. Potrebno je izpostaviti, da se model žile, ki ga ustvari in uporabi SimVascular močno razlikuje od modelov, uporabljenih v ostalih programih, model je namreč "poln", kar pomeni, da na notranji strani vsebuje satovje vozlišč in ne zgolj plašča. Za ogled rezultatov simulacije nam je ponovno priporočen ParaView (poglavje 4.2).

Žal nam program kljub obsežnemu naboru orodij ne omogoča uporabe kot knjižnica.

3.10 Končni izbor

Kljub presenetljivo širokemu naboru programov, ki se soočajo s problemom simulacije fluidov so nam kot resnejši kandidati za nadalnji razvoj ostali zgolj trije projekti: Sailfish, OpenFOAM ter Palabos. V času odločitve in izbora ogrodja, ki ga želimo uporabiti, je bil SimVascular še neobjavljen, vendar tudi če bi bil, bi zaradi svoje izjemno specifične strukture težko spremenil končno odločitev.

Največja pomanjkljivost projekta Sailfish in razlog za njegovo izločitev je opazno zamiranje njegovega razvoja ter odvisnost od enega samega razvijalca in s tem povezanega pomanjkanja za podporo. Projekt je bil sicer z razvijalcem prijaznim jezikom zelo mamljiv, vendar oznaka eksperimentalnega projekta ne navdaja s sigurnostjo, ki se jo pričakuje od ogrodja, na katerem se načrtuje še nadaljnji razvoj.

Tako sta nam preostala novi in obetavni projekt švicarske univerze Palabos ter preizkušeni in industrijsko uveljavljeni OpenFOAM. Žal v trenutnem stanju nobeden od naštetih ne nudi možnosti knjižnice, primerne za vpeljavo

v program NeckVeins. Palabos sicer kaže aktivnost na razvoju v smeri javanske ovojnice (za sicer `C++` kodo), vendar mu zaradi nezmožnosti prevoda izvorne kode ovojnice in več tem na forumu za podporo, ki so ostale neodgovorjene, to težko štejemo v prid. Tako bi komunikacija s katerokoli rešitvijo potekala preko terminala in izvajanja izvršljivih datotek. Zaradi načina delovnega toka je za takšno uporabo veliko primernejši Palabos, v katerem si lahko sami ustvarimo simulacijo, torej sprogramiramo generično simulacijo, ki potrebuje le še definicijo vhodnih in izhodnih odprtin modela, ter po želji nekaj dodatnih specifikacij lastnosti tekočine. Izbrali smo torej Palabos. Poleg naštetih razlogov, je imela velik vpliv na izbor tudi že vnaprej pripravljena simulacija anevrizme, ki se nahaja med Palabosovimi primeri uporabe.

Poglavje 4

Pregled orodij in tehnologij

4.1 Palabos

Knjižnica Palabos je programsko ogrodje za računsko reševanje dinamike fluidov, katerega jedro sloni na mrežni Boltzmannovi metodi. Njegov osnovni programski vmesnik je zelo preprost in omogoča hitro vzpostavitev simulacije fluida. Za uporabnike, ki so bolj vešči mrežne Boltzmannove metode, pa omogoča tudi lastno razširitev knjižnice z prilagojenimi modeli reševanja. V osnovi so za tri dimenzionalno simulacijo implementirani štirje numerični modeli: D3Q13, D3Q15, D3Q19 in D3Q27.

Osnovni programski jezik programskega vmesnika je `C++`. Poleg POSIX specifikacije ter MPI, knjižnica praktično nima zunanjih odvisnosti in je zato zelo preprosta za namestitev na različne sisteme, vključno s superračunalniki. Že sama po sebi podpira paralelizacijo procesov, vključno s predpripravo simulacije, kar pogosto povzroča težave drugim orodjem [10].

Čeprav knjižnica nudi ovojnico za Python in povoje ovojnice za Javo, smo se odločili za direktno uporabo `C++` vmesnika, saj se je izkazal za najbolje dokumentiranega, najstabilnejšega in veliko bolj prilagodljivega od ostalih možnosti. Za vhodne podatke, v primeru, da ne želimo programsko ustvarjati simulacijskega območja, je potrebna datoteka v obliki STL. Rezultate je iz simulacije možno zajeti ročno v poljubnih oblikah, ogrodje samo po sebi pa

podpira shranjevanje podatkov v ASCII ali binarni obliki grobih podatkov, izvoz slik v GIF obliki zapisa, za zahtevnejšo poobdelavo pa nam je na voljo tudi izvoz v formatu VTK, definiranemu s strani odprtokodne VTK knjižnice, ki je pogosto uporabljana za vizualizacijo znanstvenih podatkov.

4.2 ParaView

Paraview je izjemno fleksibilno in vsestransko orodje za analizo in vizualizacijo podatkov. Za njim stoji isto podjetje (Kitware), ki je razvilo knjižnico VTK, zato se ponuja na dlani kot najboljša rešitev za vizualizacijo podatkov, ki jih Palabos že nativno lahko izvaža. Uporabljali smo ga za ogled in analizo vseh rezultatov, ki smo jih pridobili iz simulacij s programom Palabos.

4.3 Datotečne oblike

Ker je velik del problema, ki ga rešujemo, povezava dveh večjih programov z že definiranimi vmesniki, moramo kot integrator poznati vse podatkovne strukture, ki so uporabljene v izmenjavi podatkov, in s tem del vmesnikov. Naši objekti za podatkovno izmenjavo (angl. data transfer object - DTO) so zaradi oblike vmesnikov kar datotečni zapisi.

V nadaljenju so predstavljene vse datotečne strukture, ki jih moramo razumeti za uspešno izmenjavo podatkov: format OBJ kot izhodni podatek iz programa NeckVeins, format STL kot vhodni podatek za simulacijski program, ki sloni na knjižnici Palabos, in VTK kot volumetrični zapis razultata oz. izhodni podatek simulacijskega programa.

4.3.1 Format OBJ

Wavefront OBJ[34] format zapisa podatkov je za nas pomemben, ker so nam v tej obliki na voljo rezultati orodja NeckVeins. Vsebuje 3D geometrično definicijo objekta, kar pomeni položaj vsakega oglišča, teksturne UV koordinate oglišč, normale oglišč ter ploskve, ki vsebujejo reference na tri oglišča.

Normale ploskev so definirane po pravilu zaporedja v obratni smeri urinega kazalca, zato dodatna definicija normal ploskev ni potrebna. Za nas so relevantni zgolj podatki o ogrodju objekta, torej položaji oglišč in definicije ploskev, saj le ti vplivajo na simulacijo pretoka fluida.

Podatki se v tem zapisu hranijo kot ASCII tekstovni vnosi, odvisni od ključne oznake na začetku vrstice. Pomembnejše od teh oznak so:

- `v`: koordinata oglišča (x, y, z, w), kjer je vrednost w neobvezna in preddefinirana kot 1.0
- `vt`: texturna koordinata oglišča (u, v, w), kjer so vse vrednosti na intervalu [0, 1], vrednost w je tudi tu neobvezna in je preddefinirana kot 0
- `vn`: normala oglišča (x, y, z), ki ni obvezno enotni vektor
- `f`: definicija ploskve, ki referencira uporabljena oglišča z njihovimi indexi. Lahko vsebuje tudi podatke o pripadajočih teksturnih vozliščih in normalah.

4.3.2 STL način zapisa podatkov

STL[30] je akronim za angleško besedo STereoLithography. Po slovensko stereolitografija je oblika slojnega ustvarjanja modelov, tesno z njo se uporablja izraz hitro prototipiziranje. Postopek je razvil in patentiral Chuck (Charles W.) Hull ter za namene trženja patenta v obliki 3D tiskalnikov ustanovil podjetje 3D Systems[5]. Za podporo posredovanja podatkov njihovim napravam so razvili lastno obliko zapisa podatkov tridimenzionalnih modelov, STL.

Zapis vsebuje samo podatke o trikotnih ploskvah, definiranih s tremi oglišči ter normalo, definirano z enotskim vektorjem. Obstajajo razširitve zapisa, ki omogočajo še zapis barve. Oglišča so definirana kot pozitivne kartezične koordinate v tridimenzionalnem prostoru. Vse vrednosti koordinat so

```
v  v_x  v_y  v_z  [v_w]
v  ...
...
vt  vt_u  vt_v  [vt_w]
vt  ...
...
vn  vn_x  vn_y  vn_z
vn  ...
...
f  v_1[/[vt_1][/vn_1]]  v_2[/[vt_2][/vn_2]]  v_3[/[vt_3][/vn_3]]
f  v_1  v_2  v_3
f  v_1/vt_1  v_2/vt_2  v_3/vt_3
f  v_1//vn_1  v_2//vn_2  v_3//vn_3
f  v_1/vt_1/vn_1  v_2/vt_2/vn_2  v_3/vt_3/vn_3
f  ...
...
```

Diagram 5: Način OBJ podatkovnega zapisa z natančneje razčlenjenimi vsemi možnimi načini zapisa ploskve.

števila s plavajočo vejico, podana z mantiso ter črko e, kateri sledi eksponent. Glede na normalo ploskve si sledijo po pravilu desne roke. STL zapis ne nudi ponovne uporabe vozlišč.

Poznamo dva načina zapisa STL formata: ASCII (Diagram 6) (angl. American Standard Code for Information Interchange) in binarni zapis. Struktura ASCII oblike zapisa vsebuje sledeče vrednosti:

- `solid`: poljubno polje, poimenuje objekt
- `facet normal`: zaznamuje trikotnik in poda koordinate normale trikotnika
- `endfacet`: zaznamuje konec trikotnika
- `outer loop`: zaznamuje začetek zanke, ki pa se v praksi ne uporabljajo
- `endloop`: zaznamuje konec zanke
- `vertex`: zaznamuje podatke oglišča

```
solid ime
     facet normal n_x n_y n_z
          outer loop
               vertex v1_x v1_y v1_z
               vertex v2_x v2_y v2_z
               vertex v3_x v3_y v3_z
          endloop
     endfacet
     .
     .
     .
endsolid ime
```

Diagram 6: ASCII oblika STL podatkovnega zapisa.

Ker je ASCII oblika prostorsko zelo potratna in poleg samega ASCII formata (potrebuje vsaj 7 bitov na znak) vsebuje še ogromno količino nepotrebnih podatkov, se za večje modele uporablja binarni zapis (Diagram 7). Ta poleg glave datoteke vsebuje samo še število oglišč ter podatke o ploskvah. Te so tudi v binarnem zapisu predstavljene z normalo, orientiranimi oglišči in dolžino atributov. Vsaka od vrednosti bodisi normale ali vozlišč je zapisana s standardom IEEE 754 za enojno natančnost. Pri tem morajo biti vse vrednosti oglišč nenegativne. Dolžina atributov je zapisana kot dva bajta veliko nepredznačeno število, ki pa je vedno nastavljeno na nič in se v praksi ne uporablja.

4.3.3 Format VTI

Format VTI[16] je način, ki ga uporablja Palabos za shranjevanje oz. izvoz volumetričnih podatkov. Ta oblika zapisa bazira na isti odprtokodni knjižnici

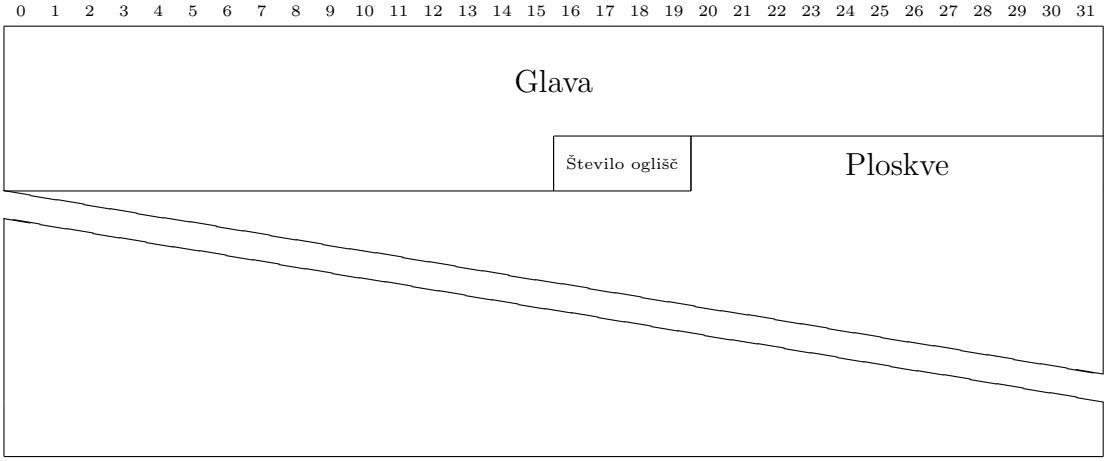


Diagram 7: Sestava STL datoteke opisane v binarnem načinu

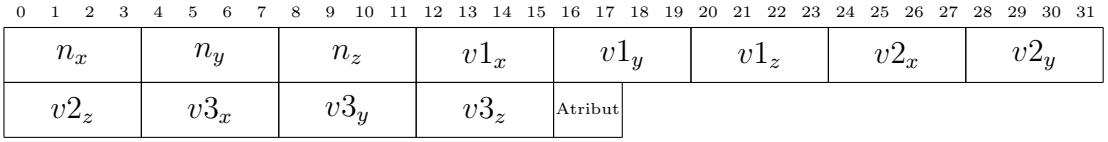


Diagram 8: Sestava ene ploskve v binarno zapisani STL datoteki

VTK kot vizualizacijski program ParaView, ki je tudi eden od peščice programov, sposobnih prikaza podatkov, shranjenih v tej obliki. VTI je eden od naslednikov opuščenega načina zapisa podatkov VTK, ki se ga kljub zastarelosti lahko izbere kot obliko izvoza podatkov simulacije. VTI zapis ohranja fleksibilnost predhodnika, vendar ne podpira izključno binarnega zapisa podatkov. Vse sodobne oblike zapisa so v osnovi XML dokumenti z v naprej definiranimi značkami, ki opisujejo, kakšni podatki so shranjeni v datoteki, in jih tako ne definira uporabljena končnica, temveč vsebina datoteke.

V primeru datoteke (diagram 9), ki ponazarja izgled rezultata simulacije z Palabos-om, tako vidimo, da imamo eno samo vsebino tipa `ImageData`. `ImageData` sicer z atributom `WholeExtent` definira celotno opazovano območje, ki je opisano z največjo in najmanjšo vrednostjo v vsaki koordinatni smeri. Točke in celice prostora so implicitno definirane z atributoma `Origin`, ki definira izvor koordinatnega sistema in `Spacing`, ki definira debelino rezin v vsaki od koordinatnih smeri.

Znotraj značke `ImageData` so vgnezdene značke `Piece`, ki glede na atri-

but `Extent` razmejujejo opazovano območje, vendar mora, ker imamo samo en `Piece` vnos, že ta obsegati celoten opazovani prostor, ki ga je definirala značka `ImageData` z atributom `WholeExtent`. Značka `PointData` v našem primeru služi zgolj kot način grupiranja `DataArray` podatkov, kjer so shranjeni nam zanimivi dejanski podatki izmerjenih vrednosti med simulacijo. Kot že omenjeno, so ti razporejeni po implicitno definirani mreži, ki jo definirajo atributi starševskih elementov.

```
<?xml version="1.0"?>
<VTKFile type="ImageData"
 version="0.1" byte_order="LittleEndian">
 <ImageData WholeExtent="x1 x2 y1 y2 z1 z2"
      Origin="x0 y0 z0" Spacing="dx dy dz">
        <Piece Extent="x1 x2 y1 y2 z1 z2">
            <PointData>
                <DataArray type="Float32" Name="variable_name"
                 format="binary" encoding="base64">
                ...
                <DataArray/>
            </PointData>
        </Piece>
    </ImageData>
</VTKFile>
```

Diagram 9: Struktura VTI datoteke, okrnjena na obliko, ki jo dobimo kot rezultati simulacije

4.4 WebGL

WebGL je na OpenGL osnovi baziran JavaScript vmesnik, ki ob predpostavki, da je podprt s strani uporabljenega brskalnika, izpostavlja nizkonivojski dostop do grafične procesne enote (GPE) računalnika elementom znotraj spletne strani. S tem imamo znotraj HTML5 objektnega modela dokumenta (Document Object Model, DOM) omogočeno strojno pospešeno računanje,

kar se uporablja za interaktivni prikaz kompleksnejših 2D in 3D elementov, brez kakršnih koli dodatkov v brskalniku.

Vmesnik je razvil in standardiziral konzorcij družbe Khronos, avtor tudi domorodnega vmesnika za izris grafike s pomočjo GPE OpenGL, na katerega različici za vgrajene sisteme (OpenGL ES 2.0) bazira WebGL. Prva standardizacija vmesnika je bila objavljena leta 2011, v izdelavi pa je tudi verzija 2.0 [13].

4.5 three.js

Za lažjo uporabo nizkonivojskega vmesnika WebGL smo uporabili visikonivijsko knjižnico three.js [24], ki nam služi kot ovojnica za WebGL. Delo nam olajša z bogato podporo različnim podatkovnim strukturam in preprostejšim, kompaktnim naborom ukazov, potrebnih za prikaz, obdelavo in interakcijo s 3D prostori in objekti.

4.6 NeckVeins / Med3D

Končni cilj rešitve, ki jo bomo razvili, je integracija simulacije pretoka tekočine po žilah v obstoječo spletno aplikacijo Med3D [4]. Začetki te aplikacije so bili v obliki javanskega programa NeckVeins za prikaz in obdelavo medicinskih podatkov o žilah, pridobljenih iz medicinskih slik. Prikazuje lahko tako kot mrežne modele, kot tudi volumetrične podatke. Aplikacija je bila v zadnjem letu prenešena v spletno obliko, ki nosi ime Med3D.

Poglavje 5

Implementacija

Kot dokaz koncepta smo si zadali integracijo izbrane knjižnice za simulacijo pretoka fluida v spletno obliko aplikacije NeckVeins. Ker žal nobena od ponujenih možnosti ne nudi rešitve v skriptnem jeziku, ki bi za komunikacijo z grafično procesno enoto uporabljal vmesnik WebGL, smo se morali zadovoljiti z oddaljeno predobdelavo modela, ki potrebne podatke sporoči strežniški enoti, ta pa nato izvede simulacijo lokalno z izvedbo vnaprej pripravljenega generičnega programa za simulacijo.

5.1 Priprava 3D modela

Modeli, ki so nam na voljo iz aplikacije NeckVeins, so iz večih razlogov neprimerni kot vhodni podatki za knjižnico Palabos. Ta pričakuje vhodne podatke v obliki zapisa STL ter, da ima model (uporabniško) definirane vhode in izhode. Med testiranjem se je prav tako izkazalo, da "osamelci", torej posamezni deli definiranega modela, ki so popolnoma ločeni od osrednjega modela in posledično od pretoka fluida, povzročajo težave pri simulaciji.

V namen simulacije smo s programom za obdelavo 3D modelov naš model ročno odprli in odstranili ploskve, kjer naj bi bile odprtine. Da bi bilo to možno narediti avtomatično, bi morali prirediti obstoječi program NeckVeins, kar bi bilo veliko bolj kompleksno.

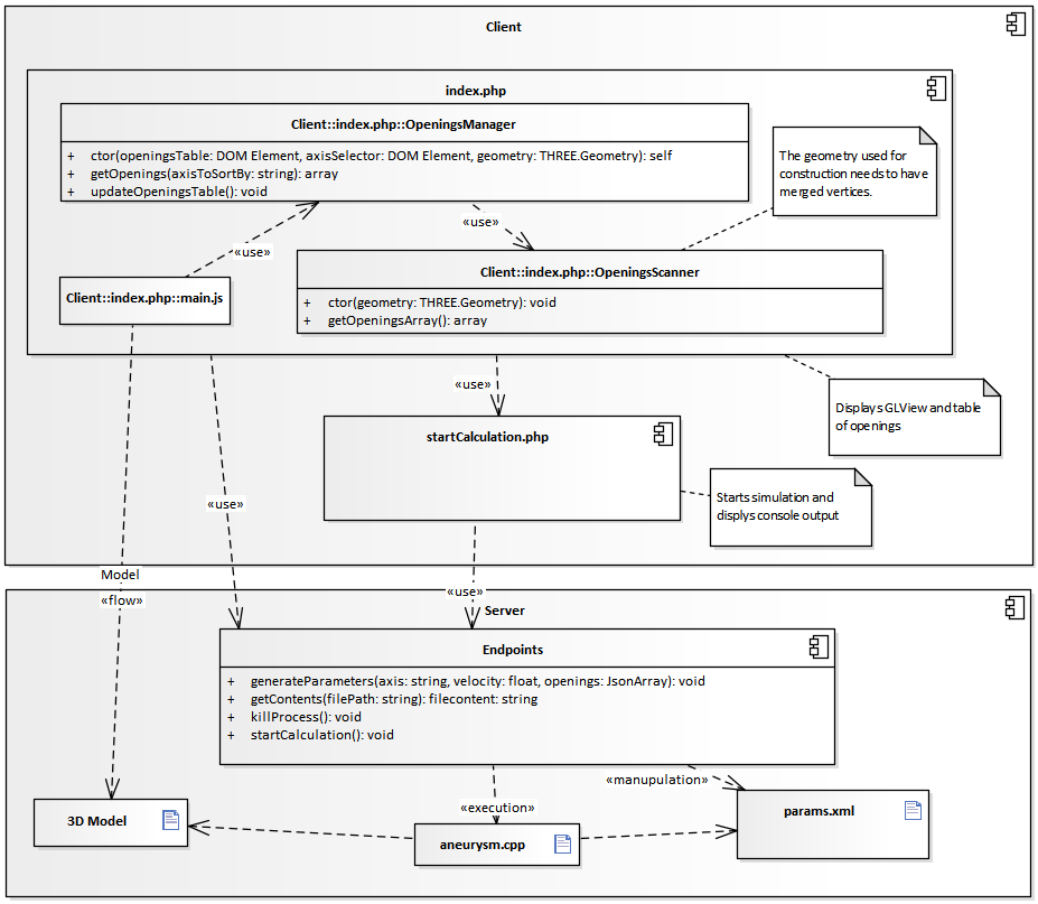


Diagram 10: Komponentni diagram rešitve

Model, ki je bil v izvornem zapisu v podatkovni obliki OBJ, smo po ročnem odpiranju odprtin izvozili v obliki STL in s tem rešili problematiko oblike zapisa modela.

5.1.1 Čiščenje modela

Za očiščenje modela smo razvili krajši program v shriptnem jeziku Python, ki prepozna največji zvezni objekt modela in iz celotnega modela odstrani dele, ki s tem modelom niso povezani. V programu najprej preberemo OBJ datotetko v podatkovno strukturo, ki jo v nadaljevanju lahko programsko obdelujemo.

V naslednjem koraku si iz optimizacijskih razlogov ustvarimo dvodimenzionalno vpogledno tabelo, ki ima dolžino enako številu vseh vozlišč modela. Druga dimenzija je variabilna in vsebuje reference (indekse) vseh ploskev, ki vsebujejo vozlišče, definirano s prvo dolžino. Ob predpostavki, da so vse ploskve, ki si delijo vsaj eno vozlišče, zvezne, tako dobimo najmanjše enote, ki so kandidati za največji zvezen objekt modela. V naslednjem koraku se sprehodimo skozi tabelo kosov in jih na podlagi preseka ploskev, ki jih enoti vsebujeta, zlivamo skupaj. Če si dve enoti delita vsaj eno skupno ploskev, ju zlijemo skupaj v eno enoto. Ta postopek ponavljamo, dokler se pri obhodih dogajajo zlivanja. Ko pridemo do točke, da se sprehodimo skozi celotno tabelo zveznih enot, in ugotovimo, da med njimi ni več nobene skupne ploskve, smo tako iz vpogledne tabele dobili tabelo vseh nepovezanih enot modela. Med njimi najdemo enoto z največjim številom ploskev in jo izberemo kot novi, očiščen model.

Na novo pridobljene podatke moramo zapisati nazaj v OBJ format, pred tem pa še počistiti nepovezana vozlišča. Zaradi načina OBJ zapisa podatkov bi bilo sicer mogoče zgolj odstraniti neuporabljene ploskve in ohraniti stara vozlišča, ki jih nobena od ploskev nebi vsebovala, vendar odstranimo iz modela tudi ta prosta vozlišča in popravimo indeksne reference ploskev na njih.

5.2 Program za simulacijo

Simulacijski program smo izdelali v izvornem jeziku knjižnice Palabos, `C++`. V grobem naš simulacijski program sledi kompleksnejšemu primeru uporabe, tj. simulacija pretoka krvi skozi anevrizmo, ki je dostavljena že z osnovno namestitvijo knjižnice.

Program kot vhodni podatek sprejme zgolj en argument, in sicer pot do konfiguracijske XML datoteke, iz katere na začetku prebere podatke, potrebne za simulacijo. Ti vključujejo:

- podatke o uporabljeni geometriji

Slika 5.1: Rezultat simulacije, prikaz pritiska znotraj žile

  - lokacijo STL modela (v istem začetnem koraku tudi preberemo sam STL model v spomin)
  - povprečno vstopno hitrost tekočine v model
  - podatke o odprtinah (definicije katere so vhodne in katere izhodne odprtine)

- podatke o simulirani tekočini
  - viskoznost
  - gostota
  - volumen

- nastavitve simulacije, med drugim:
  - maksimalen dovoljen čas simulacije (če prej ne pride do konvergence)
  - konvergenčna toleranca (epsilon)
  - ali naj se izvaja izpisovanje med simulacijo

  - ali naj se shranjujejo slike simulacije

- numerične podatke

Po prebiranju podatkov program zažene simulacijo, ki lahko, odvisno od konfiguracije, po zaključku oz. doseženi konvergenci prve simulacije preide v poljubno število dodatnih, izpopolnitvenih obhodov. Ti dosežejo natančnejšo ločljivost rezultatov simulacije in vedno temeljijo na rezultatu prejšnjega ločljivostnega nivoja.

Na začetku same simulacije ustvarimo mrežo po celotni opazovani domeni (vsebujoči škatli uporabljenega modela). Po uvozu modela uporabimo prebrane podatke o vhodnih in izhodnih odprtinah, da definiramo, kje v modelu se te nahajajo. Same odprtine modela sicer knjižnica sama zazna, potrebno jih je še primerno označiti. Za simulacijo moramo definirati tudi, katera od vozlišč želimo opazovati, torej označiti tista, ki ležijo znotraj vnešenega modela. V naslednjem koraku definiramo še lastnosti mejnih pogojev, torej odnosa delcev s ploskvami, ki jih definira model. Zadnje, kar pripravimo za simulacijo, je še sledilnik vrednosti, s katerim opazujemo, če je stanje simulacije konvergiralo k stabilnemu stanju, kar nam omogoči, da lahko simulacijo predčasno prekinemo. Stanju simulacije sledimo s pomočjo povprečne energije znotraj simulacije.

Sama simulacija poteka znotraj zanke, ki ob vsaki iteraciji preverja, če smo ali presegli dovoljen čas ali pa dosegli konvergenco energije. Izkazalo se je, da zapisovanje zajetih, predvsem volumetričnih, podatkov sredi simulacije močno vpliva na čas, porabljen za dosego konvergence. Zato je možno, in pogosto smiselno, količino in pogostost zapisovanja stanja omejiti. V končnem produktu smo želeli simulacijo predstaviti v videu z tridesetimi sličicami na sekundo, zaradi česar smo definirali shranjevanje vsakih 67 ms simulacije.

Po izpisu trenutnega stanja, če je trenutna iteracija zahtevala izpis, preverimo še stanje znotraj mreže, če je že konvergiralo. Tudi to zahteva dodatni čas, vendar v nasprotju z izpisom podatkov, ki vsebuje predvsem bralno-pisalne procese, predvsem računskega. Metoda za računanje povprečne energije je sicer veliko hitrejša kot shranjevanje podatkov, vendar lahko ob prepo-

gostem preverjanju po nepotrebnem upočasni program, zato se zadovoljimo s preverjanjem stanja vsakih dvajset iteracij.

Sledi jedro simulacije, in sicer klic za izvršitev enega koraka simulacije, sestavljenega iz računanja trkov delcev in njihovega posledičnega premika: `lattice->collideAndStream();`. Za uporabnika knjižnice je, kot vidimo, sam računski del popolnoma enkapsuliran v klic ene same metode na mreži, ki jo pripravimo. Ko se korak preračuna, se vrnemo na začetek zanke. Če smo dosegli konvergenco ali pa presegli časovni okvir, s simulacijo zaključimo in izvršimo še zadnji, bolj podroben končni zajem podatkov.

5.3 Strežniški del

Ker smo zaradi simulacije žal odvisni od domorodnega programa, ki ne more živeti znotraj spletne aplikacije, potrebujemo za našo prevedeno `C++` kodo poznano okolje, v katerem smo postavili strežnik, ki lahko naš program izvrši. Naša strežniška enota je izjemno preprosta in služi predvsem strežbi spletne strani ter zagotavljanju dostopnih točk za posredovanje uporabniško definiranih parametrov modela ter izvrševanju simulacijskega programa s posredovanimi konfiguracijskimi podatki. Strežniški del je napisan v skriptnem jeziku PHP.

Vsebuje tri enote:

- `index.php` Služi streženju osnovne spletne strani s celotno odjemalčevo JavaScript logiko za prikaz in obdelavo 3D modela

- `generateParameters.php` končič za oddajo parametrov modela

- `startCalculation.php` stran za pogon simulacije in sledenju njenega poteka, omogoča tudi predčasno končanje procesa

5.4 Del odjemalca in uporabniški vmesnik

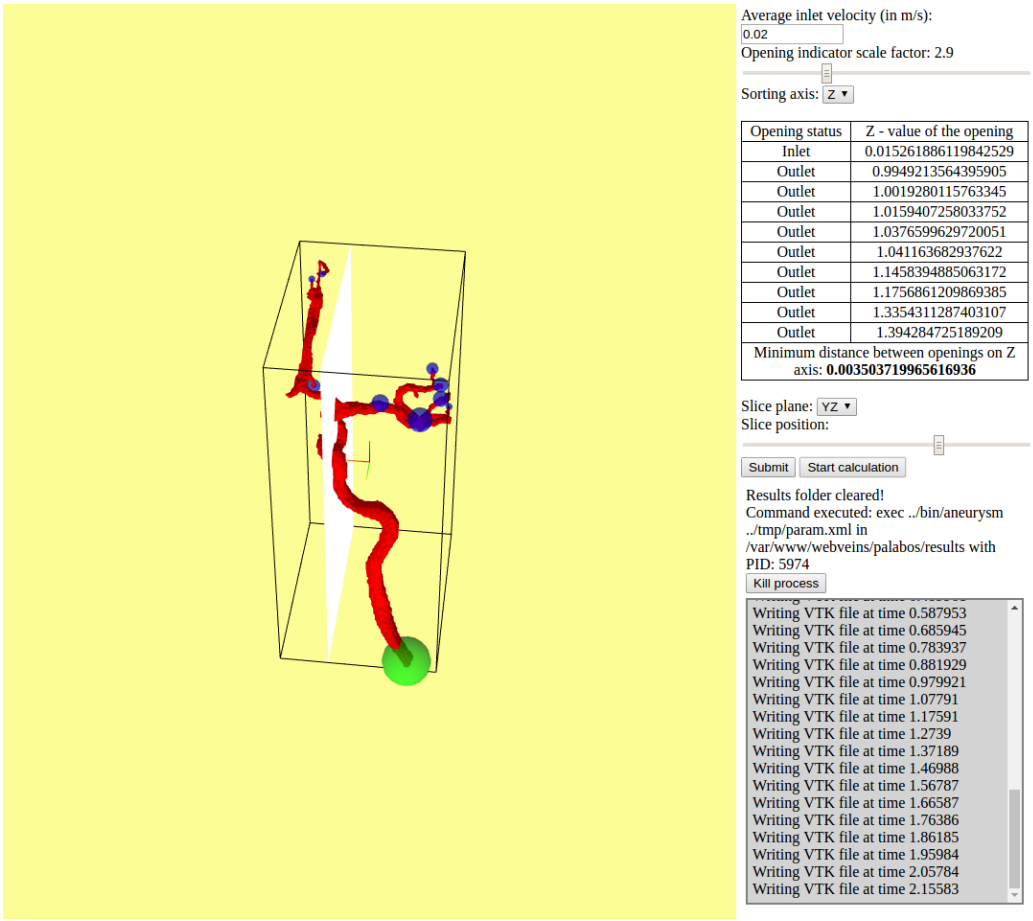
Odjemalčev program je napisan v JavaScript skriptnem jeziku, ki se izvaja v brskalniku. Namen uporabniškega vmesnika je olajšanje predprocesnih korakov, potrebnih za zagon simulacije pretoka fluida. Njegove glavne naloge so:

- prenos 3D modela s strežnika
- prikaz naloženega modela
- analiza modela za namene določanja odprtin
- prikaz in obdelava zaznanih odprtin
- izbor prereza za prikaz rezultatov

Zaradi predprocesnih korakov (predvsem odpiranje modela, pa tudi primeren podatkovni zapis), ki jih naš uporabniški vmesnik ne podpira, smo omejeni na konkreten, vnaprej izbran model, ki ga lahko uporabimo za simulacijo. Tega naš program ob odprtju spletnega naslova `./index.php` avtomatično prenese s strežnika in ga s pomočjo knjižnice three.js prikaže v okvirju, označenem kot 3D platno. Model, preden ga prikažemo, tudi analiziramo z algoritmom za iskanje odprtin. Na zaznanih odprtinah dodatno prikažemo krogle, s katerimi je možna interakcija: klik na kroglo zamenja stanje njene definicije (iz vhodne odprtine v izhodno in obratno). Za lažji pregled numeričnih podatkov prikazujemo tudi tabelo trenutnega stanja definicij odprtin, sortiranih po eni od koordinatnih osi. Ko smo z anotacijo modela zadovoljni, posredujemo stanja odprtin stražniku, ki ima s tem zadostne pogoje za izvršitev simulacije.

5.4.1 Algoritem iskanja odprtin

Pri iskanju odprtin smo se zanesli na zelo preprosto lastnost 3D mrež, ki jih sestavljajo ploskve: da lahko neko stranico proglasimo kot robno, mora



Slika 5.2: Uporabniški vmesnik

za vozlišči, ki jo opisujeta, obstajati ena, in natanko ena direktna povezava. Takoj, ko se pojavi druga povezava, predvsem če je v obratni smeri, pomeni, da je stranica zgolj stičišče dveh ploskev.

Da naša predpostavka velja, potrebujemo samo en predpogoj, in sicer, da so vozlišča ponovno uporabljena skozi celotno mrežo, in ne, da ima vsaka ploskev tri unikatna vozlišča, ki pripadajo le njej. Žal gre v obliki STL podatkovnega zapisa točno za tak primer. Zato moramo pred analizo poskrbeti, da vozlišča, ki so si dovolj blizu (ponavadi v identičnih točkah koordinatnega sistema), združimo. Na srečo nam daje ta postopek na voljo že knjižnica three.js.

Ko imamo na voljo model, ki ustreza našim pričakovanjem, si iz optimiza-

cijskih razlogov vnaprej pripravimo oz. ustvarimo tabelo končne dolžine. Ta nam bo služila za memoizacijo povezav med vozlišči. V naslednjem koraku se sprehodimo skozi vse ploskve modela in v memoizacijsko tabelo vnašamo povezave med oglišči ploskve (a-b, b-a, b-c, c-b, c-a, a-c). Končni produkt je tabela, ki za vsakega od vozlišč pozna njegova sosednja povezana vozlišča. Pomembno v tej tabeli pa je, da morajo iz prej navedenega razloga biti *vsa* sosednja vozlišča v tabeli povezav *dvakrat*. Če kje temu ni tako, smo našli robno vozlišče.

Za vsakega od vozlišč tako preverimo, če je robno. Če je, rekurzivno sledimo njegovim enojnim robnim povezavam, dokler se ne vrnemo nazaj v izhodiščno vozlišče. Skupek vozlišč, ki jih na tak način obhodimo, proglasimo za odprtino.

Ker je za simulacijo potreben podatek predvsem lokacija odprtine na eni od osi, moramo določiti lokacijo kot eno točko v prostoru. Za določitev lokacije se zadovoljimo s povprečno vrednostjo vseh vozlišč odprtine, saj točne lokacije ne potrebujemo. Za prikaz indikatorjev odprtin na modelu je potrebno poznati tudi velikost odprtine, za kar vzamemo razdaljo od v tem koraku izračunane pozicije odprtine do najbolj oddaljenega vozlišča odprtine. Potencialna izboljšava te določitve bi bila lahko določitev radija odprtine z očrtano kroglo okoli oblike, ki jo opisuje odprtina.

Naš postopek ima tudi dve omejitvi:

- "Obrnjene" stranice (z obratno orientacijo trikotnikov) ne zazna kot potencialno odprtino. Odvisno od razumevanja modela, je to lahko tudi pričakovano obnašanje

- Obstaja robni primer, ko algoritem, v primeru, da se dve odprtini stikata v enem od robnih vozlišč, obe proglasi za eno samo. Vzrok za to je način iskanja stranic odprtine, ki dovoljuje iz enega vozlišča več kot dve možni robni stranici.

5.4.2 Anotiranje odprtin

Uporabniški vmesnik nam kot indikatorje zaznanih odprtin prikaže krogle na modelu, ki imajo središče v središču odprtine, in so sorazmerno z velikostjo odprtine tudi velike. Za pomoč pri prikazovanju zelo majhnih odprtin imamo na voljo tudi drsnik, ki nam po potrebi omogoča prilagajanje velikosti indikatorjev odprtin. Zelene krogle prikazujejo vhodne odprtine, medtem ko modre prikazujejo izhodne. Vrsto odprtine lahko spreminjamo s klikom na njen indikator. Vzporedno s spremembo barve indikatorja se posodobi tudi tabela vseh odprtin.

Naš program za opis vrst odprtin vzame zgolj zaporedje vrst odprtin in po kateri koordinatni osi je to zaporedje opisano. V ta namen v tabeli vseh odprtin prikazujemo tudi vrednosti odprtin na izbrani koordinatni osi. S pogledom na tabelo lahko dodatno zagotovimo pravilen opis odprtin tako, da izberemo tisto os, na kateri so vhodne odprtine čimbolj oddaljene od izhodnih. S tem zmanjšamo možnost, da sta položaja dveh odprtin, ki sta zelo blizu, zaradi različnega načina zaznave, zamenjana.

5.4.3 Anotiranje prereza za analizo

Ker celotni rezultat simulacije lahko obsega več deset gigabajtov volumetričnih podatkov, smo se za prikaz rezultatov v uporabniškem vmesniku namesto volumetričnih podatkov odločili uporabiti zgolj določene prereze prostora, ki ga model žile obsega. V uporabniškem vmesniku imamo prikazan mejni prostor modela, znotraj katerega lahko izberemo ploskev na eno od treh ravnin, ki so pravokotne na koordinatne osi: ravnine XY, XZ in YZ. Ravnino, ki jo izberemo, vidimo na predogledu skupaj z modelom žile in jo lahko z drsnikom pomikamo vzdolž koordinatne osi, na katero je pravokotna. Na tej ploskvi lahko na koncu simulacije projeciramo GIF sliko, ki je poleg volumetričnih VTK podatkov tudi možen način izvoza podatkov med simulacijo in ob njenem zaključku.

5.4.4 Zagon simulacije

Ko na žili opravimo vse potrebne anotacije, lahko zaženemo simulacijo. V uporabniškem vmesniku se nam v prostoru pod elementi, ki smo jih uporabljali v prejšnjih korakih, prične izpisovati izhodni tok procesa, ki se izvaja asinhrono na strežniku. Prav tako nam je voljo tudi gumb za prekinitev simulacije, s katerim lahko proces zaključimo ročno še preden je simulacija dosegla konvergenco ali časovno omejitev. To se zgodi s pošiljanjem signala SIGTERM, ki ga program ujame in se sam primerno zaključi.

```
1: function FindOpenings(model)          ▷ Model is a mesh with joined vertexes
2:     vertexConnections[model.vertices.size]
3:     for each face ∈ model do
4:         a ← face.vertices[0]                    ▷ a, b, c, are vertex indices
5:         b ← face.vertices[1]
6:         c ← face.vertices[2]
7:         append b, c to vertexConnections[a]
8:         append a, c to vertexConnections[b]
9:         append a, b to vertexConnections[c]
10:    end foreach

11:    edgeVertices[ ]
12:    for i ∈ {0, ..., vertexConnections.size} do
13:        if i not in any of edgeVertices[ ] then
14:            evs ← FindConnectedEdgeVertices(i, vertexConnections, [ ])
15:            append evs to edgeVertices
16:        end if
17:    end for
18:    return b
19: end function

20: function FindConnectedEdgeVertices(index, connections, foundVertices)
21:    if index in foundVertices then
22:        return foundVertices
23:    end if

24:    append index to foundVertices
25:    connectingVertices ← connections[index]

26:    for each vertex ∈ connectingVertices do unique
27:        if vertex is unique in connectingVertices then
28:            foundVertices ← FindConnectedEdgeVertices(vertex, connections, foundVertices)
29:        end if
30:    end foreach
31: end function
```

Diagram 11: Iskanje odprtin modela

Poglavje 6

Zaključek

V diplomski nalogi smo raziskali široko in raznoliko področje knjižnic za simulacijo dinamike fluidov. Izbrali smo takšno, ki je odprtokodna, zastonjska in se jo da vpeti v avtomatiziran proces. Obstoječe podatke iz programa NeckVeins smo deloma priredili, da so pripravljeni za avtomatizirano uporabo, deloma pa smo razvili algoritma za avtomatizacijo in pomoč pri ročni pripravi simulacije.

Predelali smo program za simulacijo pretoka krvi skozi predel žile, kjer se nahaja anevrizma, in za pomoč pri zagonu simulacije razvili spletni uporabniški vmesnik. V vmesniku lahko vidimo zgolj okrnjeno obliko rezultatov simulacije v obliki slik, čeprav je na strežniku na voljo ogromna količina volumetričnih podatkov.

Z našo rešitvijo žal še tesneje povežemo odjemalca s strežnikom, kar pa je nujna cena za sprejemljivo hitrost izvajanja simulacije [17], saj brskalniki še niso dovolj močni, da bi lahko podpirali tako zahtevne procese. Kljub temu bi bil zanimiv izziv poizkusiti obsežno knjižnico Palabos, ali pa zgolj naš simulacijski program, s pomočjo prevajalnika Emscripten prevesti v asm.js obliko JavaScript skriptnega jezika. Za to bi bilo seveda potrebno program primerno prirediti.

Za integracijo simulacije pretoka krvi po žili v program Med3D bi bilo v spletni uporabniški vmesnik potrebno tudi dodati rešitev za korake, ki

smo jih mi morali na modelu izvesti ročno v urejevalniku tridimenzionalnih objektov. To vključuje prevod in po potrebi optimizacijo skripte za čiščenje osamelcev iz osrednjega modela, odpiranje končičev žile, ki jih za simulacijo označujemo kot vhodne in izhodne odprtine, ter zapiranje lukenj v žili, ki se lahko pojavljajo na neprimernih delih modela. V tem koraku bi bilo potrebno tudi avtomatično zaznati realno velikost modela, saj se merske enote, ki so lahko prisotne v OBJ obliki zapisa modela, s prevodom v STL, ki je potreben za simulacijo, izgubijo.

Zanimivo bi bilo prikazati tudi volumetrično končno stanje simulacije, morda celo stanja med samo simulacijo. Za to bi bila morda primerna uporaba v zadnjem letu razvite knjižnice vtk.js, ali pa celo lastna implementacija prikaza s pomočjo knjižnice three.js.

Literatura

[1] ANSYS CFD. Dostopno na http://www.ansys.com/Products/Fluids/ANSYS-CFD.

[2] Autodesk CFD. Dostopno na http://www.autodesk.com/products/cfd/overview.

[3] James Maxwell Buick. *Lattice Boltzmann Methods in Interfacial Wave Modelling*. Doktorska dizertacija, University of Edinburgh, 1997. Dostopno na https://web.archive.org/web/20070902011040/http://www-personal.une.edu.au/~jbuick/Publications/PDF/Thesis/tot.html.

[4] Matija Marolt Ciril Bohak, Primož Lavric. Med3D - Spletno vizualizacijsko ogrodje volumetričnih medicinskih podatkov s podporo oddaljenemu sodelovanju. *Delavnica Elektronsko in mobilno zdravje: zbornik 19. mednarodne multikonference Informacijska družba - IS 2016, 10.-11. oktober 2016, Ljubljana, Slovenija*, Zvezek G:14 – 16, 2016.

[5] 3D Systems Corporation. 3D Systems Founder Chuck Hull to be Inducted into the National Inventors Hall of Fame. Dostopno na http://www.3dsystems.com/press-releases/3d-systems-founder-chuck-hull-be-inducted-national-inventors-hall-fame.

[6] FELiScE - Finite Elements for LIfe SCiences and Engineering. Dostopno na https://gforge.inria.fr/projects/felisce/.

[7] FEniCS. Dostopno na http://fenicsproject.org/.

[8] FlowKit. Dostopno na http://www.flowkit.com/.

[9] FlowKit. Lattice Boltzmann Method - the kernel of Palabos. Dostopno na http://www.palabos.org/software/lattice-boltzmann-method.

[10] FlowKit. Palabos - Introduction. Dostopno na http://www.palabos.org/documentation/userguide/introduction.html.

[11] U. Frisch, B. Hasslacher, in Y. Pomeau. Lattice-Gas Automata for the Navier-Stokes Equation. *Phys. Rev. Lett.*, 56:1505–1508, Apr 1986. Dostopno na http://link.aps.org/doi/10.1103/PhysRevLett.56.1505.

[12] Martin Gardner. Mathematical Games – The fantastic combinations of John Conway's new solitaire game "life", 10 1970.

[13] Khronos Group. WebGL 2 Specification. Editor's Draft Tue Jan 17 18:14:10 2017 -0800. Dostopno na https://www.khronos.org/registry/webgl/specs/latest/2.0/.

[14] Thomas H. Pulliam Harvard Lomax in David W. Zingg. Fundamentals of Computational Fluid Dynamics, 1999.

[15] Inria. Dostopno na http://www.inria.fr/en/.

[16] Kitware. File Formats for VTK Version 4.2. Dostopno na http://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf.

[17] Primož Lavrič. Spletno ogrodje za vizualizacijo volumetričnih podatkov z možnostjo oddaljenega sodelovanja. 2016. Dostopno na https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=85027.

[18] Aljaž Maslo. *Numerično modeliranje razlitja nafte v reko z uporabo mrežne Boltzmannove metode.* Doktorska dizertacija, Fakulteta za gradbeništvo in geodezijo, 2015. Dostopno na https://repozitorij.uni-lj.si/IzpisGradiva.php?id=32532&lang=slv.

[19] A. A. Mohamad. *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*. Springer, 2011 edition, 8 2014.

[20] National Program for Applications-Oriented Research in CFD. Dostopno na `http://www.grc.nasa.gov/WWW/wind/index.html`.

[21] OpenFOAM. Dostopno na `http://www.openfoam.com/`.

[22] Palabos. Dostopno na `http://www.palabos.org/`.

[23] GitHub repozitorij za Palabos. Dostopno na `https://github.com/gladk/palabos`.

[24] Odprtokodni projekt: lastnik repozitorija Ricardo Cabello (mrdoob). WebGL 2 Specification. Editor's Draft Tue Jan 17 18:14:10 2017 -0800. Dostopno na `https://threejs.org/`.

[25] REO. Dostopno na `https://team.inria.fr/reo/`.

[26] Sailfish. Dostopno na `http://sailfish.us.edu.pl/index.html`.

[27] SCOTT SHEPPARD. Project Falcon graduates from Autodesk Labs to Autodesk Flow Design. Dostopno na `http://labs.blogs.com/its_alive_in_the_lab/2014/01/project-falcon-graduates-from-autodesk-labs-to-autodesk-flow-design.html`.

[28] SimVascular. Dostopno na `http://simvascular.github.io/`.

[29] Solidworks Flow. Dostopno na `https://www.solidworks.com/sw/products/simulation/flow-simulation.htm`.

[30] STL. Dostopno na `http://www.fabbers.com/tech/STL_Format`.

[31] Sauro Succi. *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond (Numerical Mathematics and Scientific Computation)*. Oxford University Press, reprint edition, 7 2013. Dostopno na `http://amazon.com/o/ASIN/019967924X/`.

[32] Projekt Unicorn. Dostopno na https://launchpad.net/unicorn.

[33] Kompatibilnost projektov Unicorn in FEniCS. Dostopno na https://answers.launchpad.net/unicorn/+question/264528.

[34] Wavefront. Obj Specification. Dostopno na http://www.martinreddy.net/gfx/3d/OBJ.spec.

[35] Wind-US. Dostopno na http://www.grc.nasa.gov/WWW/winddocs/index.html.

[36] Obtaining Wind-US. Dostopno na http://www.grc.nasa.gov/WWW/winddocs/install/obtaining.html.

[37] X-Flow Homepage. Dostopno na http://www.xflowcfd.com/index.php.